91602
new 82-10

COMPLIMENTARY COPY
Please Credit Ontario Hydro Research if
Reproduced in Whole or in Part

ontario hydro
research division

To: Dr. D. Mills
Director of Research

SOME STATISTICAL ASPECTS OF THE COMPUTERIZED RANDOM TESTING OF CRITICAL SOFTWARE

Three statistical aspects of the computerized random testing for critical software are discussed. First, two random number generators are described. Second, the effect of dependence in the estimation of the software reliability is analyzed. It was found that for positive correlated sequence of trials, the sample size to obtain a given confidence level in the reliability estimates is larger than the one needed for an independent sequence. Finally, the effect of partition testing is analyzed. It was found that sampling in partitions with the same probability of trial failure is equivalent to random sampling. A methodology is also given to analyze other situations.

1. INTRODUCTION

Darlington Nuclear Generation Station (DNGS) uses a computerized shutdown system (SDS) to initiate a reactor trip in case of specific accidents. Since this is a safety feature, it is necessary to test that the software complies with its performance requirements and that it meets its specified reliability targets. The testing is carried out by simulating many reactor conditions that may arise during the normal operation of the reactor and during specific accidents and comparing the outcome of the SDS system with the expected one. If there is any discrepancy, it is necessary to make the adequate modifications to correct the problem and to repeat the tests. If there is no discrepancy, a statement of the confidence levels at which compliance is achieved is issued.

This report deals with three statistical aspects of software reliability testing. These are (i) the effect or impact of the random number generator; (ii) the effect of dependency in the trials; and (iii) the effect of partition of the sampling space during testing. This work was requested by Mr. G. H. Archinoff of the Safety Systems Unit, Nuclear Safety Department, Design & Development Division - Generation.

2. RANDOM NUMBER GENERATORS

Two random number generators have been investigated: the congruential shift-register, RSUPER, of Marsaglia, *et al.*, (1973); and the RANDOM of Wichman and Hill (1982). Both provide adequate methodology to generate random numbers and exhaustive tests have been performed to ensure adequacy of the generated values. The RSUPER method has a period of about 5×10^{18} and the RANDOM a period of 2.78×10^{13}, so that generating 1000 random numbers per second continuously, the sequence would not repeat over 800 years.

job	file	date	report no.
72-165	809.7	July 19, 1991	91-107-K

Both algorithms require that certain values (seeds) be initiated before the first entry to the program. By keeping track of these numbers, it is possible to regenerate the same sequence when required. The FORTRAN code for both methods, together with testing cases, is given in Appendix 1.

3. DEPENDENCE IN THE RANDOM TESTING

3.1 Markov Model Order 1

In order to examine the effects of dependency among successive trials during the software testing, different models for the dependency are proposed. The simplest case is when only two successive tests are correlated. This corresponds to a Markov model of first order. Other cases are analyzed in the next section. To analyze the first order Markov model, it is assumed that the unreliability of the software system is q (*i.e.*, the unconditional probability of having a successful trial is $p = 1 - q$). If ρ is the amount of correlation, then it can be shown that the probability of having a success in a trial, given that the previous trial was also a success, is

$$P(S|S) = p + \rho q.$$

This result can now be used to estimate upper confidence intervals for q or for estimating the required sample size for given reliability and confidence levels. If only successes are observed during the testing, the model that relates n, q, and the confidence $1-\alpha$ is

$$p\,(p + \rho q)^{n-1} = \alpha.$$

For given n, the value of p that estimates the reliability with confidence $1-\alpha$ is given by the solution of the above equation. Non-analytical methods are needed to solve it. For given confidence $1-\alpha$ and reliability p, the required sample size is given by

$$n = 1 + (\log(\alpha) - \log(p))/\log(p - \rho q).$$

Table 1 provides a comparison of the sample size required to estimate $p = 1 - 10^{-4}$ with 50% confidence for $\rho = -1\ (0.05)\ 1$. The column $P(s|S)$ gives the conditional probability of having a success after observing one; N gives the required sample size under dependency; and N/No is the ratio between N and the nominal sample size (under no dependency), No. A plot of N/No vs. ρ is given in Figure 1. As can be seen, the required sample size for the dependent case ranges from 50% less than the nominal sample (when $\rho = -1.0$) to 100 times larger (when $\rho = 0.99$). Further analyses have found that the ratio N/No is rather constant for different values of confidence and reliability.

3.2 Higher Markov Models

Higher order Markov models can also be proposed for the dependency of the trials during the software testing. The m-order Markov model assumes that the outcome of trial i given all the previous trials, is equivalent to the outcome of trial i given the outcome of only trials $i-1$, $i-2$, ... $i-m$. In this case, it is necessary to specify 2^{m+1} probabilities, corresponding to all possible success/failure sequences on $m+1$ trials. To completely specify them, it is necessary to provide the same number of restrictions. In practice, only the first and second order moments of the probabilities are specified, which provide $1+(m+1)+ m(m+1)/2$ number of constraints. These are not sufficient to fully specify the probability values.

To overcome the above difficulty, the following approach could be used which is based on the fact that the multivariate normal distribution is fully specified by the first two moments. First, a success on trial i is associated with the event $(Z_i < z_p)$, where Z_i is a standard normal random variate, z_p is the

$1 - p$ percentile of the normal distribution, and the joint distribution of the Z_i's follows a markov process of order m. The second step is to assign the dependence structure on the normal random variables. This is done by setting the correlation between Z_i and Z_j so that

$$P\{Z_i<z_p \ \& \ Z_j<z_p\} = p^2 + \rho_{ij}pq$$

where ρ_{ij} is the correlation between the trials i and j.

The probability of observing n consecutive successes in the software testing is given by

$$P\{S_1, \ldots, S_m, \ldots, S_n\} = P\{S_1, \ldots, S_m\} \prod P\{S_i|S_{i-1}, \ldots, S_{i-m}\}.$$

Because the S_i's form a markov chain of order m,

$$P\{S_i|S_{i-1}, \ldots, S_{i-m}\} = P\{S_{m+1}|S_m, \ldots, S_1\}$$
$$= P\{S_{m+1}, S_m, \ldots, S_1\}/P\{S_m, \ldots, S_1\}$$

The above terms can be evaluated using the equivalence with the normal events, *i.e.*,

$$P\{S_m, \ldots, S_1\} = P\{Z_m<z_p, \ldots, Z_1<z_p\}.$$

The program MULNOR (Schervish, 1984) can be used to evaluate these probabilities.

To estimate the required sample size for obtaining the reliability p with confidence $1-\alpha$, the solution to the following equation is needed

$$\alpha = P\{S_1, \ldots, S_m\} P\{S_{m+1}|S_1, \ldots, S_m\}^{n-m}.$$

Table 2 provides results for the sample size required to estimate $p = 1 - 10^{-4}$ with 50% confidence for Markov models of order 1 to 3. The CORRELATION SUMMARY indicates the relationship between the correlation among trials and the correlation of the normal variables for the given value of unreliability. For example, to obtain a nominal 10% correlation between the trials would require a 70% (0.7030 in the normal column) correlation between the normal variables. On the other hand, a nominal 10% correlation between the normal variates would produce a 0.6% (0.5960E-03 in the binomial column) correlation between the trials.

The results for the Markov model of order 1 are similar to those given in Table 1. The differences are due to the numerical approximations used in the calculation of the probabilities. The correlations for the Markov models order 2 and 3 are given in the BINOMIAL AUTOCOVARIANCE columns (the correlation between trial i and trial $i+k$ are given in the k^{th}column). The results indicate that for correlations up to 50% among three consecutive trials, the testing requires up to 3.5 times the number of independent trials to estimate p with the same confidence.

4. PARTITION TESTING: PARTITION EFFECT

This section presents a discussion of the methodology that can be used to calculate the $1-\alpha$ upper confidence estimate for the probability that the software will fail in a given instance. The basic assumption is that the software domain can be partitioned into L subdomains and that testing is independently carried out in each one of them. Two different cases are considered. The first assumes equal probabilities of trial failure in all subdomains. The second assumes different probabilities of trial failure in the subdomains.

4.1 Subdomains with the Same Probability of Failure

Suppose the software domain is partitioned into L subdomains, and the probability of trial failure in each subdomain is q, the same for all of them. Then, the overall probability of trial failure is also given by q. Now, suppose that n_i independent observations are taken from subdomain i, i=1,...,L and that no failures are observed. The 1-α confidence estimate is given by the value q_e such that

$$\text{Prob(no failure in the sample} \mid n_i \text{ obs for dom}_i,\ i=1,\ldots,L;\ q_e) = \alpha.$$

Under the assumption that all subdomains have the same probability of trial failure, the conditional probability is given by $(1-q_e)^n$, where $n = \Sigma n_i$. So, q_e is given by

$$q_e = 1 - (\alpha)^{1/n}. \tag{4.1}$$

It is interesting to note that the above expression is exactly the same as the one used to estimate q when a random sample of size n is taken. This is the case because of the assumption that all the subdomains have the same probability of trial failure.

Hamlet and Taylor (1990) propose the use of the estimate $h = 1 - (1 - p)^L$ for software reliability. As they pointed out, this estimate is more stringent than mere operational reliability (*ie.*, the probability that the software would fail during its application). In fact, 1 - h represents the probability that the software is *defect-free* under the assumption that the subdomains are independent. This assumption is in contradiction with the assumption that the subdomains form a partition (*ie.*, they are mutually exclusive and exhaustive) since exclusive events are not independent. Therefore, it seems that *partition testing* in Hamel and Taylor has a meaning which is not clearly specified in their article and which is different from the one that has been used here.

4.2 Subdomains with Different Probability of Failure

Suppose the software is partitioned into L subdomains, and that the probability of trial failure in subdomain i is q_i, i=1,...,L, not necessarily the same for all of them. The reliability of the software is given by 1-q, where q is the probability of trial failure. The value of q is given by

$$\begin{aligned} q &= \Sigma\ P(\text{dom}_i)P(\text{Software failure}|\text{dom}_i) \\ &= \Sigma\ \pi_i q_i \end{aligned} \tag{4.2}$$

where π_i is the probability of using subdomain i.

Now, suppose that n_i independent observations (trials) are taken from subdomain i and that no failures are observed. Then, the 1-α confidence estimate is given by the value q_d such that

$$\text{Prob(no failure in the sample} \mid n_i \text{ obs from dom}_i,\ i=1,\ldots,L;\ q_d) = \alpha.$$

The conditional probability is given by $(1-q_1)^{n1}(1-q_2)^{n2}\ldots(1-q_L)^{nL}$. Therefore, the equation to solve is

$$\alpha = (1-q_1)^{n1}(1-q_2)^{n2}\ldots(1-q_L)^{nL}.$$

The above equation has an infinite number of solutions for q_i, i=1,...,L. Each solution set provides joint 1-α estimates for the reliability of the subdomains, and by using equation 4.2, it provides an estimate for q. To select a single estimate, it is necessary to impose further restrictions. There are no a priori rules, but one could select the solution set for which

$$r_{i1} = q_i/q_1 = (1 - \alpha^{1/ni})/(1- \alpha^{1/n1}).$$

This choice would select the estimates for the q_i's in the same ratio as the *independent* estimates for the un-reliability of the subdomains (this would be the optimal choice if it is assumed that the sample size is allocated inversely proportional to the probability of software failure in the subdomains). Other selections are possible as well. For example, the r_{i1} could be set to 1 (*ie.*, all q_i's assumed equal). In this case, the solution simplifies to the one given in equation 4.1.

5. CONCLUSION

The FORTRAN code for two random number generators have been provided. By keeping track of the seed values, it is possible to repeat the sequence of numbers, as well as to continue the testing after interruptions. This feature should provide more repeatability to the testing.

The analysis of the dependence effect in the software testing indicates that the relation between the independent and dependent sample sizes needed to estimate the reliability with a given reliability depends on the level of dependence. For a first order Markov dependency with negative correlation, the sample size is smaller than that obtained in the independent case. For a first order Markov and positive correlation, the sample size is larger than that obtained in the independent case, and for large correlation values, sample size is substantially larger. Large sample increases are also observed when the dependency follows high order Markov models with positive correlations.

The analysis of the effect of partition testing indicates that if the same probability of trial failure is assumed for all the subdomains, then the partition testing is equivalent to random testing. A methodology was given to study the effect of partition testing when different subdomains have different trial failure probabilities.

Approved:

A.H. Chung
Manager
Operations Research Department

Submitted:

F. Camacho
Mathematician
Statistical Methods & Analysis Unit
Operations Research Department

FC:cmh

REFERENCES

1. Hamlet, D., and Taylor, R., (1990). "Partition Testing Does not Inspire Confidence". *IEEE Trans. on Soft. Eng.*, 16, pp. 1402-1411.

2. Marsaglia, G., Ananthanarayasan, K., and Paul, N., (1973). "How to Use the McGill Random Number Package Super-duper". School of Computer Science, McGill University.

3. McLeod, A.I., (1982). "Efficient Fortran Coding of a Random Number Generator". Technical Report TR-82-08, Department of Statistical & Actuarial Sciences, The University of Western Ontario.

4. McLeod, A.I., (1985). "A Remark on Algorithm AS 183. An Efficient and Portable Pseudo-random Number Generator". *Applied Statistics*, 34, pp. 198-200.

5. Schervish, M.J., (1984). "Algorithm AS 195: Multivariate Normal Probabilities with Error Bound". *Applied Statistics*, 33, pp 81-94.

6. Wichmann, B.A., and Hill, I.D., (1982). "Algorithm AS 183: An Efficient and Portable Pseudo-random Number Generator". *Applied Statistics*, 31, pp 188-190.

TABLE 1

Required Sample Size Under a First Order Markov Model

Relib	0.000100
Conf	0.5
No	6931.125

Rho	p(s|S)	N	N/No
-1.00	0.99980	3465.889	0.500
-0.95	0.99981	3554.741	0.513
-0.90	0.99981	3648.270	0.526
-0.85	0.99982	3746.854	0.541
-0.80	0.99982	3850.915	0.556
-0.75	0.99983	3960.923	0.571
-0.70	0.99983	4077.401	0.588
-0.65	0.99984	4200.939	0.606
-0.60	0.99984	4332.198	0.625
-0.55	0.99985	4471.925	0.645
-0.50	0.99985	4620.967	0.667
-0.45	0.99986	4780.289	0.690
-0.40	0.99986	4950.990	0.714
-0.35	0.99987	5134.336	0.741
-0.30	0.99987	5331.785	0.769
-0.25	0.99988	5545.030	0.800
-0.20	0.99988	5776.046	0.833
-0.15	0.99989	6027.150	0.870
-0.10	0.99989	6301.082	0.909
-0.05	0.99990	6601.102	0.952
0.00	0.99990	6931.125	1.000
0.05	0.99991	7295.886	1.053
0.10	0.99991	7701.177	1.111
0.15	0.99992	8154.149	1.176
0.20	0.99992	8663.743	1.250
0.25	0.99993	9241.282	1.333
0.30	0.99993	9901.327	1.429
0.35	0.99994	10662.91	1.538
0.40	0.99994	11551.43	1.667
0.45	0.99995	12601.51	1.818
0.50	0.99995	13861.59	2.000
0.55	0.99996	15401.70	2.222
0.60	0.99996	17326.83	2.500
0.65	0.99997	19802.00	2.857
0.70	0.99997	23102.22	3.333
0.75	0.99998	27722.54	4.000
0.80	0.99998	34653.01	5.000
0.85	0.99999	46203.79	6.666
0.90	0.99999	69305.37	9.999
0.95	1.00000	138610.0	19.998
0.99	1.00000	693047.8	99.991

TABLE 2

Required Sample Size Under Higher Order Markov Models

Confiability	50%
Un-reliability	0.0001
Tolerance Error	0.1000E-04

Correlation Summary

NOMINAL	BINOMIAL	NORMAL
0.0000	0.0000	0.0000
0.1000	0.5960E-03	0.7030
0.2000	0.1192E-02	0.8093
0.3000	0.4172E-02	0.8713
0.4000	0.1073E-01	0.9134
0.5000	0.2324E-01	0.9434

Results for Markov Model Order 1

REQSIZ	NOMSIZ	CHANGE	BINOMIAL AUTOCOVARIANCES
6930	6930	1.0000	0.0000
7696	6930	1.1105	0.1000
8665	6930	1.2504	0.2000
9905	6930	1.4293	0.3000
11547	6930	1.6662	0.4000
13859	6930	1.9999	0.5000

Results for Markov Model Order 2

REQSIZ	NOMSIZ	CHANGE	BINOMIAL	AUTOCOVARIANCES
6885	6930	0.9935	0.0000	0.0000
7676	6930	1.1076	0.1000	0.0000
8330	6930	1.2020	0.1000	0.1000
9077	6930	1.3098	0.2000	0.1000
10033	6930	1.4478	0.2000	0.2000
10033	6930	1.4478	0.3000	0.1000
10857	6930	1.5667	0.3000	0.2000
12150	6930	1.7532	0.3000	0.3000
11468	6930	1.6548	0.4000	0.1000
12075	6930	1.7424	0.4000	0.2000
13244	6930	1.9111	0.4000	0.3000
15081	6930	2.1762	0.4000	0.4000
13745	6930	1.9834	0.5000	0.2000
14682	6930	2.1186	0.5000	0.3000
16517	6930	2.3834	0.5000	0.4000
19251	6930	2.7779	0.5000	0.5000

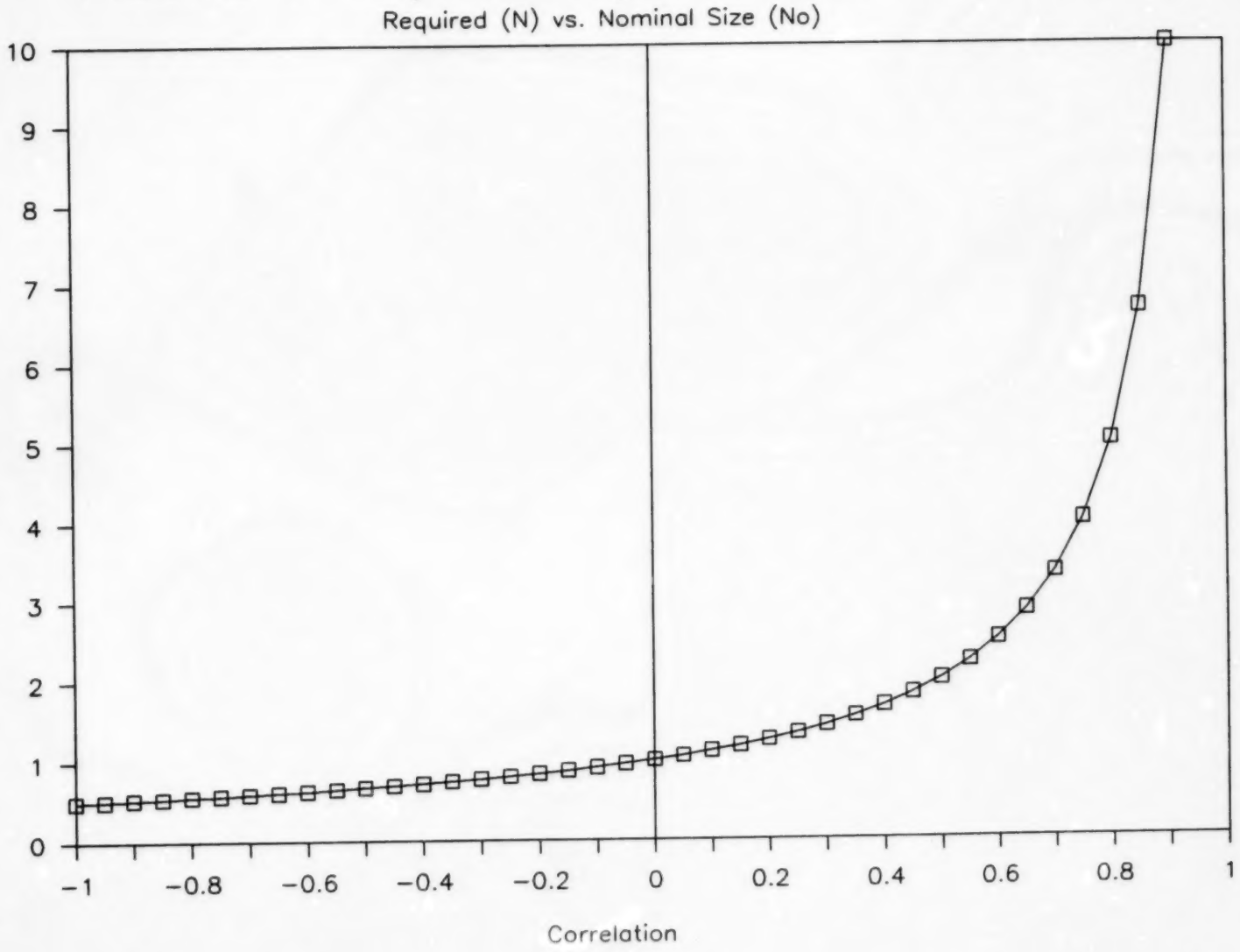
TABLE 2 (Cont'd)

Required Sample Size Under Higher Order Markov Models

Results for Markov Model Order 3

REQSIZ	NOMSIZ	CHANGE	BINOMIAL AUTOCOVARIANCES		
6942	6930	1.0017	0.0000	0.0000	0.0000
8945	6930	1.2908	0.1000	0.1000	0.1000
9618	6930	1.3879	0.2000	0.1000	0.1000
10373	6930	1.4968	0.2000	0.2000	0.1000
11300	6930	1.6306	0.2000	0.2000	0.2000
10600	6930	1.5296	0.3000	0.1000	0.1000
11181	6930	1.6134	0.3000	0.2000	0.1000
12188	6930	1.7587	0.3000	0.2000	0.2000
12383	6930	1.7869	0.3000	0.3000	0.1000
13094	6930	1.8895	0.3000	0.3000	0.2000
14480	6930	2.0895	0.3000	0.3000	0.3000
12214	6930	1.7625	0.4000	0.2000	0.1000
13274	6930	1.9154	0.4000	0.2000	0.2000
13319	6930	1.9219	0.4000	0.3000	0.1000
13992	6930	2.0190	0.4000	0.3000	0.2000
15320	6930	2.2107	0.4000	0.3000	0.3000
15524	6930	2.2401	0.4000	0.4000	0.2000
16611	6930	2.3970	0.4000	0.4000	0.3000
18723	6930	2.7017	0.4000	0.4000	0.4000
14850	6930	2.1429	0.5000	0.3000	0.1000
15401	6930	2.2224	0.5000	0.3000	0.2000
16999	6930	2.4530	0.5000	0.3000	0.3000
16779	6930	2.4212	0.5000	0.4000	0.2000
17538	6930	2.5307	0.5000	0.4000	0.3000
19775	6930	2.8535	0.5000	0.4000	0.4000
19842	6930	2.8632	0.5000	0.5000	0.3000
21652	6930	3.1244	0.5000	0.5000	0.4000
24530	6930	3.5397	0.5000	0.5000	0.5000

FIGURE 1: Dependency in Random Testing



APPENDIX 1

Random Number Generators

1. RSUPER

The RSUPER algorithm is based in a combination of the congruential and the shift register generators. The congruential generator used is

$$x_i = 69069 x_{i-1} \bmod 2^{32}.$$

The initial seed should be an odd positive number. The shift generator generates random bits using

$$b_j = b_{j-17} + b_{32i+j}.$$

Each set of 32 bits provides the binary representation of an integer $(y_{i-1})_{BIN} = b_{32i}\ b_{32i-1} \cdots b_{32(i-1)}$. The bit representation $(y_0)_{BIN} = b_{32}b_{31}...b_1$ of an initial seed, y_0, provides the start values for the recurrence. Finally, the outputs x_i and y_j are combined by bit-wise addition modulo 2 (or bit-wise exclusive-or) to obtain the random variable z_i as

$$z_i = (x_i \oplus y_i + 1)/(2^{32}+1).$$

The code presented here is based on those given by McLeod (1982).

The routine requires two seeds to start the iteration. The first should be an odd integer, and there is no restriction for the second.

To test the program, initializing Iseed = 1 and Jseed = 1 after 1000 iterations, the following numbers should be obtained: Iseed = 1636533665, Jseed = 4153239278 (or -141728018 as a signed integer in 32 bits machines) and Rsuper = 0.5860426O.

```
C-----------------------------------------------------------------------
        Function    RSUPER(Junk)
C-----------------------------------------------------------------------
C RSUPER.
C IBM-PC    for  Microsoft    Fortran
C
        integer*4     Iseed,    Jseed,    it
        integer*4     i1,  i2,  i3
        Equivalence       (Rit,   it),   (ri2,i2),     (rjseed,     itemp),     (rit1,iit1)
        Common       /ASUPER/     iseed,    jseed,    it
        Data         RMULT/2.3283064359967937E-10/
C
C Mask32   has  1's  in  the  rightmost    31  bits  and  0 in  the  leftmost    bit
C
         Data         Mask32    /8#17777777777/
C
C Generate    Random    Integer    Using   Congruential      Generator
C
         Iseed    = 69069*Iseed
C
C Generate    Random    Integer    Using   Shift   Register    Generator
         I1   = ISHFT(Jseed,      -15)
         I2   = IEOR(I1,     Jseed)
         I3   = ISHFT(I2,     17)
         Jseed    = IEOR(I2,     I3)
C
C Combine    and  Convert    to  Uniform(0,1)
         IT   = IEOR(Iseed,      Jseed)    + 1
         Rsuper    = 0.0
         If  (IT   .le.   0)  Rsuper    = 0.5
         IT   = IAND(It,     Mask32)
         Rsuper    = Rsuper    + Float(IT)*Rmult
         Return
         End
```

2. RANDOM

The RANDOM algorithm is based on three multiplicative congruential generators, each using a prime number for its modulus (30269, 30307, 30323) and a primitive root (171, 172, 170) for its multiplier. The random number is the fractional part of the sum of the three numbers. The code presented here is the one given by Wichmann and Hill, modified according the suggestion given by McLeod (1985).

The routine require three integers (numbers between 1 and 30000) to be selected as seeds for the algorithm. To test the program, initializing IX = 1, IY = 1, and IZ = 1, after 1000 iterations should produce IX = 9292, IY = 22099, IZ = 27520 and RANDOM = 0.94371410.

```
C-----------------------------------------------------------------------
      FUNCTION    RANDOM(L)
C-----------------------------------------------------------------------
C          ALGORITHM     AS  183    APPL.   STATIST.    (1982)   VOL.31,    NO.2
C          RETURNS    A  PSEUDO-RANDOM      NUMBER   RECTANGULARLY      DISTRIBUTED
C          BETWEEN    0  AND   1
C-----------------------------------------------------------------------
C
C          IX,   IY  AND   IZ  SHOULD    BE  SET   TO  INTEGER    VALUES    BETWEEN
C          1  AND   30000    BEFORE    FIRST    ENTRY
C
C          INTEGER    ARITHMETIC     UP  TO  30323    IS  REQUIRED
C
      COMMON    /RAND/    IX,   IY,   IZ
      IX  =  171   *  MOD(IX,    177)   -  2   *  (IX   /  177)
      IY  =  172   *  MOD(IY,    176)   -  35  *  (IY   /  176)
      IZ  =  170   *  MOD(IZ,    178)   -  63  *  (IZ   /  178)
C
      IF  (IX   .LT.   0)  IX  =  IX  +  30269
      IF  (IY   .LT.   0)  IY  =  IY  +  30307
      IF  (IZ   .LT.   0)  IZ  =  IZ  +  30323
C
C          IF  INTEGER    ARITHMETIC     UP  TO  5212632    IS  AVAILABLE,
C          THE   PRECEDING     6  STATEMENTS      MAY   BE   REPLACED     BY
C
C          IX  =  MOD(171    *  IX,   30269)
C          IY  =  MOD(172    *  IY,   30307)
C          IZ  =  MOD(170    *  IZ,   30323)
C
C          ON   SOME    MACHINES,     THIS    MAY   SLIGHTLY     INCREASE
C          THE   SPEED.    THE   RESULTS     WILL   BE   IDENTICAL.
C
      RANDOM    =  AMOD(FLOAT(IX)       /  30269.0    +  FLOAT(IY)     /  30307.0    +
     *                  FLOAT(IZ)     /  30323.0,     1.0)
      IF  (RANDOM     .GT.   0.0)   RETURN
      RANDOM    =  DMOD(DBLE(FLOAT(IX))/30269.000            +
     *                  DBLE(FLOAT(IY))/30307.000            +
     *                  DBLE(FLOAT(IZ))/30323.000             , 1.0D)
      IF  (RANDOM     .GE.   1.0)    RANDOM    =  0.999999
      RETURN
      END
```